$1.50

Apple Assembly Line

Volume 3 -- Issue 7 April, 1983

In This Issue...

Patch DOS 3.3 for Fast LOAD and BLOAD. 2
An "ORG" Macro for Self-Aligning Code. 10
New S-C Cross Reference Utility. 12
Date Processing Modules. 13
The Apple][Circuit Description (Review). 20
New Version of DOS -- Patchers Beware. 23
PATCHER: A General-Purpose Patch Installer 24
More About the PRAWM Board 28

New Goodies

We have several new products available this month. There are descriptions inside this issue of the new Cross Reference program for the S-C Macro Assembler, and the new book "Apple][Circuit Description". Also, the long-awaited RCA 1802 Cross Assembler is now ready, at $32.50.

Version 1.1 of the Macro Assembler is now ready to go! The upgrade from the current Version 1.0 will only cost you $12.50. That gets you //e, Videx, and STB 80-column support, 5 new directives and all the other new features described last month.

DISASM and the //e

Yesterday afternoon I received two phone call in less than 30 minutes, both reporting that RAK-Ware's disassembler, DISASM, does not work on the Apple //e. The problem occurs when DISASM calls an odd entry into the monitor HOME routine. At several places in the routines to enter address information Bob Kovacs used $FC5A for a sort of combination VTAB and Clear-to-End-of-Page. Well, that won't work on a //e. The following patches change all the calls to $FC5A into $FC58, or the standard HOME routine. This will change the behavior of the program a little, making the screen clear between entries, rather than just tab down, but the program should now work.

```
84C:58      94D:58      A79:58
AD8:58      BBA:58      BFB:58
```

Patch DOS 3.3 for Fast LOAD and BLOAD.....Bob Sander-Cederlof

There must be at least a dozen products on the market now to speed up DOS 3.3: Diversi-DOS, David-DOS, The DOS Enhancer, QuickDOS, FastDOS, Hyper-DOS, et cetera. Some of these are unfortunately not compatible with the everyday programs we like to use, such as the S-C Assembler, ES-CAPE, or our favorite word processor. And it can be quite difficult sometimes to determine the degree of compatibity.

For the record, S&H Software's DOS Enhancer is completely compatible with the S-C Macro Assembler. David-DOS works well until you try to use the .TF directive.

Most of the speed-up systems only improve the speed of LOAD, BLOAD, RUN, BRUN, SAVE, and BSAVE. Some also speed up booting into the language card. And two (Diversi-DOS and David-DOS) speed up READing and WRITE-ing TEXT files, as well as offering a lot of minor enhancements in pursuit of more "user-friendliness".

It seems that the more the speed-up system does, the more compatibility problems you can expect. After all, to add a feature you do have to change some code. And many programs on the market expect the DOS image to be un-modified so they can jump into DOS subroutines in strange unexpected places and make their own custom patches to the DOS image.

Paul Schlyter (a subscriber in Sweden) sent me a small patch for DOS 3.3 early in April, 1982. Paul's patch speeds up only RUN, BRUN, LOAD and BLOAD, but it such a small patch that it will almost fit into the interstices (unused bytes) inside DOS. In fact, after I removed one bug and reorganized the code a little, I was able to fit it entirely within two unused areas:
** $BA69-BA95 and $BCDF-BCFF. I believe the result is completely compatible with all the programs I use around here, except for the ones that use their own modified and protected DOS.

Paul's patch turns out to be functionally equivalent to the much longer patch proposed in HardCore Magazine's HyperDOS, but it leaves the INIT command intact.

I ran some timing tests:

LOAD	40 sectors	standard	10 sec
		patched	3.5 sec
BLOAD	37 sectors	standard	11 sec
		patched	4 sec
LOAD	132 sectors	standard	32 seconds
		patched	7.5 seconds

I didn't try measuring times, but I suspect that SAVE and BSAVE may be just a little faster with this patch installed (during the read-after-write phase).

** Not true! See "New Version of DOS -- Patchers Beware".

S-C Macro Assembler (the best there is!).....$80.00
Source code of S-C Assembler II, Version 4.0, on disk.....$95.00
Fully commented, easy to understand and modify to your own tastes.
S-C Macro Assembler /// $100.00
Preliminary version. Call or write for details.

S-C Cross Reference Utility $20.00
S-C Cross Reference Utility with Complete Source Code.....$50.00

S-C Word Processor.....$50.00
As is, with fully commented source code. Needs S-C Macro Assembler.
Applesoft Source Code on Disk.....$50.00
Very heavily commented. Requires Applesoft and S-C Assembler.
ES-CAPE: Extended S-C Applesoft Program Editor.....$60.00

AAL Quarterly Disks.....each $15.00
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983		

Double Precision Floating Point for Applesoft.....$50.00
Provides 21-digit precision for Applesoft programs.
Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)..... $79.00
Source Code for FLASH! Runtime Package.....$39.00
Full Screen Editor for S-C Macro Assembler (Laumer Research).....$49.00

The Visible Computer: 6502 (Software Masters).....	(reg. $50.00)	$45.00
Super Disk Copy III (Sensible Software).....	(reg. $30.00)	$27.00
Amper-Magic (Anthro-Digital).....	(reg. $75.00)	$67.50
Amper-Magic Volume 2 (Anthro-Digital).....	(reg. $35.00)	$30.00
Quick-Trace (Anthro-Digital).....	(reg. $50.00)	$45.00
DISASM Dis-Assembler (Rak-Ware).....		$30.00

Blank Diskettes (with hub rings).....package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks.....$36.00
Vinyl disk pages, 6"x8.5", hold one disk each.....10 for $6.00
Reload your own NEC PC-8023 ribbon cartridges.....each ribbon $5.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges....each $2.50
Diskette Mailing Protectors.....10-99: 40 cents each
100 or more: 25 cents each
ZIF Game Socket Extender.....$20.00
Ashby Shift-Key Mod.....$15.00
Lower-Case Display Encoder ROM.....$25.00
Only Revision level 7 or later Apples.

Books, Books, Books.....compare our discount prices!

"The Apple][Circuit Description", Gayler.....	($22.95)	$21.00
"Enhancing Your Apple II, vol. 1", Lancaster.....	($15.95)	$15.00
"Incredible Secret Money Machine", Lancaster.....	($7.95)	$7.50
"Micro Cookbook, vol. 1", Lancaster.....	($15.95)	$15.00
"Beneath Apple DOS", Worth & Lechner.....	($19.95)	$18.00
"Bag of Tricks", Worth & Lechner, with diskette.....	($39.95)	$36.00
"Apple Graphics & Arcade Game Design", Stanton.....	($19.95)	$18.00
"Assembly Lines: The Book", Roger Wagner.....	($19.95)	$18.00
"What's Where in the Apple", Second Edition.....	($24.95)	$23.00
"What's Where Guide" (updates first edition).....	($9.95)	$9.00
"6502 Assembly Language Programming", Leventhal.....	($16.99)	$16.00
"6502 Subroutines", Leventhal.....	($12.99)	$12.00
"MICRO on the Apple--1", includes diskette.....	($24.95)	$23.00
"MICRO on the Apple--2", includes diskette.....	($24.95)	$23.00
"MICRO on the Apple--3", includes diskette.....	($24.95)	$23.00

Add $1 per book for US postage. Foreign orders add postage needed.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We take Master Charge, VISA and American Express ***

Since the S-C Assemblers use the LOAD command to process .IN directives, large assemblies with large included files will assemble about three times faster when you install this speed-up patch.

The patch is really rather simple. But before examining the patch, let's review the normal flow inside DOS for LOADing and BLOADing.

DOS is constructed in three layers: the outer layer accepts your commands from the keyboard or from your program. The inner layer, called RWTS, handles the intimate details of reading or writing a specified sector on a specified track. RWTS also does the raw disk initialization when you use the INIT command. The layer between commands and RWTS is called the File Manager (FM).

The command layer calls FM to open, close, rename, lock, unlock, verify, or delete a file; to print a catalog; to initialize a disk; or to position within a file. There are also four kinds of calls for reading and writing files, to read or write one byte or a range of bytes.

When you use the RUN or LOAD command, the command layer calls FM to read the first two bytes. These bytes contain the length of your program. For Integer BASIC or S-C Assembler source files, the length is subtracted from HIMEM to get a loading address. The loading address for Applesoft programs is found in $67,68. Then FM is called to read a range of bytes of that length, to be stored starting at the loading address just determined.

When you use the BRUN or BLOAD command, the first four bytes are read off the front of the file. The first two bytes are the loading address, and the next two are the length. (Of course, you can override the loading address with the "A" parameter after the file name.)

After winding our way through the front end of FM, we finally get to this subroutine (where the range is read):

```
                    1000 READ.RANGE
AC96- 20 B5 B1      1010        JSR DECR.TEST.LENGTH
AC99- 20 A8 AC      1020        JSR READ.BYTE
AC9C- 48            1030        PHA                 SAVE THE BYTE
AC9D- 20 A2 B1      1040        JSR GET.ADDRESS.INC
ACA0- A0 00         1050        LDY #0
ACA2- 68            1060        PLA                 GET THE BYTE
ACA3- 91 42         1070        STA ($42),Y         STORE IN
BUFFER
ACA5- 4C 96 AC      1080        JMP READ.RANGE
```

The subroutine DECR.TEST.LENGTH breaks out of this loop when the range has been completely read. The READ.BYTE subroutine picks bytes out of the DOS buffer, and reads a sector into that buffer when the buffer is empty.

To understand the speed-up patch, break the reading process into three parts: the first sector, the last sector, and all the in-between sectors. We will let the loop shown above handle the first sector and possibly the last sector, and read the in-between sectors using a faster method. Short files with only one or two data sectors will not have any in-between sectors, and so there will be no improvement in speed.

First we need to read the rest of the first sector of the file. The first two or four bytes were already read to get address and length information. We can let the loop shown above do that job. But we need a way to break into the loop when it is our turn. Let's patch the JMP on the last line to jump to our patch.

Our patch will get control after the loop above has read and stored a byte of data. At that time our patch can look at the current file position in $B5E6; if $B5E6 is non-zero, then there are still bytes in the DOS buffer. As long as there are bytes in the DOS buffer, we will branch back to $AC96 and let FM handle the bytes in its normal way.

Once the first sector has been read and stored, a byte at a time, $B5E6 will have a zero value. Then our patch can look at the remaining length. If the remainging length is at least one whole sector, we can read it faster. If not, FM can read the last partial sector in its normal fashion.

To read a sector faster, we bypass the DOS buffer. We can temporarily patch the actual destination address where the sector must go into the RWTS call block. RWTS can put the entire sector directly into its final destination, rather than into the DOS buffer to be later moved by the rather slow loop above.

The extra time saved by eliminating the middle man will save an entire revolution of the drive to get the next sector (if it is in the same track, and they usually are). A 40 sector file laid out sequentially on three tracks will save 38 revolutions of the disk. The disk spins at 5 revolutions per second, so we will save a hair over 7 seconds. (If the file is not laid out sequentially, the savings will be less.)

The bigger the file, the bigger the percentage improvement. We can save 3 seconds per track. It normally takes FM about 18 revolutions to read a track; with our patch, a track can be read in about 3 revolutions. We save 15 revolutions or 3 seconds on each full track. That is, a full track can be read in .6 seconds instead of 3.6 seconds. The rest of the time required to read the file is spent moving the head from track to track, and reading the catalog and VTOC sectors.

If all 16 sectors of a track are to be read, and if the sectors were allocated the normal DOS 3.3 way, I think this is the way it happens with my patch installed:

```
F    E  D  C  B  A  9  8    7  6  5  4  3  2  1    0
F 0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F 0
```

The bottom line of numbers shows the physical sector numbers. As you move across the page from left to right, you simulate the disk read head. It may take up to a full revolution of the disk before sector F appears, but once it does we proceed to pick off approximately every other sector as they come by. The top line of numbers shows the DOS 3.3 logical sector numbers. Logical sector E is actually physical sector 2, and so on. So it takes two full revolutions, plus two more sectors, to read all 16.

If you are trying to figure out where the rest of the time is used, keep in mind that DOS first reads the VTOC (track 17, sector 0); then the first catalog sector (track 17, sector 15); if the file specified is not in the first catalog sector, it reads another; and so on. If the file is far down in the catalog, it might have to read all 15 catalog sectors to find the file. Then the track/sector list is read; it is usually in sector 15 of the same track containing the first 15 sectors of data. On the other hand, as the disk fills up the sectors get splattered all over the disk.

Here is the patch code, arranged so that it squeezes into those two interstices I mentioned earlier:

```
                        1000 *--------------------------------
                        1010 *    S.FAST LOAD
                        1020 *
                        1030 *       FAST "LOAD" AND "BLOAD"
                        1040 *
                        1050 *       INSTALLED IN UNUSED AREAS IN DOS 3.3:
                        1060 *           $BA69-$BA95    (45 BYTES FREE)
                        1070 *           $BCDF-$BCFF    (33 BYTES FREE)
                        1080 *--------------------------------
AC96-                   1090 READ.RANGE          .EQ $AC96
B0B6-                   1100 READ.NEXT.SECTOR    .EQ $B0B6
B36F-                   1110 END.OF.DATA.ERROR   .EQ $B36F
B5C1-                   1120 RANGE.LENGTH        .EQ $B5C1,C2
B5C3-                   1130 RANGE.ADDRESS       .EQ $B5C3,C4
B5CB-                   1140 BUFFER.ADDRESS      .EQ $B5CB,CC
B5E4-                   1150 SECTOR.COUNT        .EQ $B5E4,E5
B5E6-                   1160 BYTE.OFFSET         .EQ $B5E6
                        1170 *--------------------------------
                        1180         .OR $BA69
                        1190         .TF B.PATCH1
                        1200
BA69- AD E6 B5          1210 PATCH1 LDA BYTE.OFFSET       LAST BYTE OF
BA6C- D0 24             1220         BNE GO.READ.RANGE    A SECTOR?
BA6E- AD C2 B5          1230         LDA RANGE.LENGTH+1   WHOLE SECTOR LEFT?
BA71- F0 1F             1240         BEQ GO.READ.RANGE    NO.
BA73- AD CB B5          1250         LDA BUFFER.ADDRESS   SAVE BUFFER ADDRESS
BA76- 48                1260         PHA
BA77- AD CC B5          1270         LDA BUFFER.ADDRESS+1
BA7A- 48                1280         PHA
BA7B- AD C3 B5          1290         LDA RANGE.ADDRESS    READ DIRECTLY
BA7E- 8D CB B5          1300         STA BUFFER.ADDRESS   INTO RANGE
BA81- AD C4 B5          1310         LDA RANGE.ADDRESS+1
BA84- 8D CC B5          1320         STA BUFFER.ADDRESS+1
                        1330
                        1340 READ.LOOP
BA87- 20 B6 B0          1350         JSR READ.NEXT.SECTOR
BA8A- B0 03             1360         BCS .1
BA8C- 4C DF BC          1370         JMP PATCH2
BA8F- 4C 6F B3          1380 .1      JMP END.OF.DATA.ERROR
                        1390
                        1400 GO.READ.RANGE
BA92- 4C 96 AC          1410         JMP READ.RANGE
                        1420 *--------------------------------
```

```
                    1430          .OR $BCDF
                    1440 ,        .TF B.PATCH2
                    1450
BCDF- EE E4 B5      1460 PATCH2   INC SECTOR.COUNT
BCE2- D0 03         1470          BNE .1
BCE4- EE E5 B5      1480          INC SECTOR.COUNT+1
BCE7- EE C4 B5      1490 .1       INC RANGE.ADDRESS+1   NEXT PAGE
BCEA- EE CC B5      1500          INC BUFFER.ADDRESS+1
BCED- CE C2 B5      1510          DEC RANGE.LENGTH+1
BCF0- D0 0B         1520          BNE .2
BCF2- 68            1530          PLA                   RESTORE BUFFER
BCF3- 8D CC B5      1540          STA BUFFER.ADDRESS+1
BCF6- 68            1550          PLA
BCF7- 8D CB B5      1560          STA BUFFER.ADDRESS
BCFA- 4C 96 AC      1570          JMP READ.RANGE    ONE BYTE AT A TIME
                    1580
BCFD- 4C 87 BA      1590 .2       JMP READ.LOOP
                    1600 *--------------------------------
```

To install the patches, you need to BLOAD PATCH1 and BLOAD PATCH2. Then patch locations $ACA6-7 to 69 BA, to change the JMP READ.RANGE instruction to a JMP PATCH1. Note that you must BLOAD the patches before changing $ACA6-7. If you change $ACA6-7 first, the system will crash as soon as you try to execute a BLOAD.

Here is an Applesoft program (which you could append to your HELLO program) to poke the patches into DOS.

```
20000  REM INSTALL FAST DOS LOAD AND BLOAD PATCHES
20010  READ N: IF N = 0 THEN  END
20020  READ A
20030  FOR I = 1 TO N: READ P: POKE A,P:A = A + 1: NEXT
20040  GOTO 20010
20100  DATA  44,47721,173,230,181,208,36,173,194,181,
       240,31,173,203,181,72,173,204,181,72,173,195,
       181,141,203,181,173,196,181,141,204,181,32,
       182,176,176,3,76,223,188,76,111,179,76,150,172
20110  DATA  33,48351,238,228,181,208,3,238,229,181,
       238,196,181,238,204,181,206,194,181,208,11,
       104,141,204,181,104,141,203,181,76,150,172,
       76,135,186
20120  DATA  2,44198,105,186
20130  DATA  0
```

Paul mentioned he was working on an equally simple patch to speed up SAVE and BSAVE, but I haven't heard any more from him on that subject.

FLASH! Compiler note:

The FLASH! Integer Basic Compiler was recently reviewed by PEELINGS magazine and received an A+. It is currently the highest rated Integer compiler (the competition is rated only A). The price? Just $79.00 ($70 less than the competition)!

FASTDRAW 1.1

A software package from CASTLE DESIGNS which provides enhanced Hi-Resolution Graphics instructions for use in Applesoft Basic programs. For example ....

&DRAW,1	Erases, moves and re-draws an entire array of shapes with a single instruction.
&HPLOT,1	Erases, moves and re-plots an array of points.
&HGR,c	Performs a hi-speed screen erase to any color.

Plotting rate exceeds 4,000 points per second.

Other FASTDRAW 1.1 instructions provide scrolling, as well as drawing or plotting without erasing.

CASTLE DESIGNS FASTDRAW 1.1 ..... $29.95

This software may be used with the APPLIED ENGINEERING A/D BOARD to acquire and plot analog data. The A/D BOARD will acquire 8 channels of data with 8-bit resolution, and is very easy to use. Simply use a PEEK statement to read the data.

APPLIED ENGINEERING A/D BOARD $129.00

To acquire data with real-time accuracy, use the APPLIED ENGINEERING TIMEMASTER real time clock board in combination with the A/D BOARD and the FASTDRAW 1.1 software. The TIMEMASTER will provide timing for data acquisition as often as once per millisecond. For example, 200 data points can be acquired and plotted in 250 milliseconds. The TIMEMASTER has battery backup and an on-board timer which can measure time intervals of up to 48 days to a resolution of one millisecond.

APPLIED ENGINEERING TIMEMASTER $129.00

Included on the FASTDRAW 1.1 diskette are several programs which demonstrate its various uses and a utility which merges two shape tables. Requires Applesoft and DOS 3.3, and an APPLE II, II+, or IIe with 48K Ram.

Texas Residents Add 5% Sales Tax (Hardware Only).
Add $10.00 If Outside U.S.A.

Order any of the above items from APPLIED ENGINEERING, P.O. BOX 470301, DALLAS, TEXAS 75247 ; Phone (214) 492-2027

Or order FASTDRAW 1.1 from CASTLE DESIGNS, 2717 TEAKWOOD, PLANO, TEXAS 75075

An "ORG" Macro for Self-Aligning Code.....Bob Sander-Cederlof

Roger Johnson (Minnesota) called a week or so ago with a plea for an easy way to make program segments align themselves automatically on page boundaries. He was writing a system to be burned into EPROM and run on another computer; it would be easier to debug in the target machine if subroutines and data blocks began on even page boundaries. There was ample room, so the wasted bytes between routines didn't bother him.

Of course, the .OR directive in the S-C Macro Assembler can easily change the origin whenever you wish, but it also changes the target address (.TA directive) or closes any open target file (.TF directive). Therefore a different approach is required.

Bill Morgan and Mike Laumer described how to do this in these pages a few months back, using the .BS directive to reserve enough bytes to reach the next page boundary. But with the help of a simple macro, we can not only make it easier to make self-aligning code: we can also make it generate error messages if the origin we try to set involves backing up over a longer-than-expected predecessor.

Here is the macro definition, and a few lines demonstrating how to call the macro:

```
1000         .MA ORG
1010         .DO *>]1
1020  !!! ERROR: ORG ]1 RANGE CROSSED !!!
1030         .ELSE
1040         .BS ]1-*
1050         .FIN
1060         .EM
1070 *--------------------------------
1080         .OR $800
1090 SAMPLE LDA $1234
1100         RTS
1110         >ORG $900
1120         STA $1234
1130         RTS
1140         >ORG $980
1150 DATA    .DA #1,#2,#3
```

Line 1110 calls the ORG macro with a parameter of "$900". This means that everywhere you find "]1" in the macro definition, the assembler will see "$900". The conditional (.DO) on line 1010 will read ".DO *>$900". Since * equals $804 at this point, it is not greater than $900. Therefore the condition is false, and the lines following line 1010 will be skipped up to line 1030 where there is an ".ELSE". The lines after 1030 through the ".FIN" on line 1050 will be assembled. Line 1040 will be assembled as ".BS $900-*", which will bump the location up to $900.

Here's how the above example assembles:

```
                    1000         .MA ORG
                    1010         .DO *>]1
                    1020  !!! ERROR: ORG ]1 RANGE CROSSED !!!
                    1030         .ELSE
                    1040         .BS ]1-*
                    1050         .FIN
                    1060         .EM
                    1070 *-------------------------------
                    1080         .OR $800
0800- AD 34 12      1090 SAMPLE LDA $1234
0803- 60            1100         RTS
0804-               1110         >ORG $900
                    0000>         .DO *>$900
                    0000>         .ELSE
0804-               0000>         .BS $900-*
                    0000>         .FIN
0900- 8D 34 12      1120         STA $1234
0903- 60            1130         RTS
0904-               1140         >ORG $980
                    0000>         .DO *>$980
                    0000>         .ELSE
0904-               0000>         .BS $980-*
                    0000>         .FIN
0980- 01 02 03      1150 DATA    .DA #1,#2,#3
```

If we had written line 1110 as ">ORG $800" the condition on line 1010 would be true, causing line 1020 to be assembled. Line 1020 is illegal syntax for the assembler, so it will be listed after an error message. The "]1" will be filled in to make the line list like this:

```
*** BAD OPCODE ERROR
 1110>  !!! ERROR: ORG $800 RANGE CROSSED !!!
```

That will occur during pass one of the assembly, so no code will be generated. The error message will be the only output.

New S-C Cross Reference Utility.....Mike Laumer

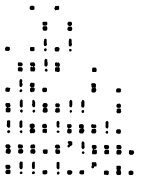
At last a Cross Reference Utility is available for the S-C Assemblers that is fully compatible with the latest releases of the S-C Macro Assembler. It handles all the new directives, shows macro calls, and can even give an optional cross reference on the opcodes! It only takes a few seconds to cross reference even a huge file and begin the listing! It is even faster than the Macro Assembler in processing the source lines.

The Cross Reference Utility also can optionally print a paginated source file listing before printing the cross reference. That way you can be certain that you have a program listing with the same line numbers shown in the cross reference listing.

The price is reasonable: only $20.00 for the object code version and $50.00 for both source and object code. What other company sells source code to their utilities!

VAGABONDO ENTERPRISES
1300 E ALGONQUIN - 3G
SCHAUMBURG IL - 60195

(312) 397-8705



FIRE ORGAN

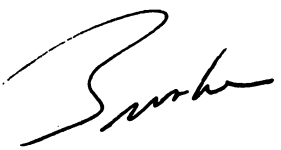
DEAR BOB:

OVER THE PAST FEW YEARS I HAVE ALWAYS ENJOYED READING AAL AND FREQUENTLY FOUND USEFUL INFORMATION. I APPRECIATE THIS TYPE OF PUBLICATION AND HOPE THAT YOU WILL BE ABLE TO CONTINUE ITS PUBLICATION INDEFINITELY. IN MY OPINION, THIS IS THE 'BEDROCK' SUPPORT LEVEL FOR PROGRAMMING THE APPLE.

I HAVE INCLUDED A CONTRIBUTION FOR AAL THAT I HAVE USED WITH A STATISTICAL ANALYSIS SYSTEM (NEVER MARKETED). IT'S FOR 'DATE' PROCESSING (NOT 'DATA'), AND AN OVERVIEW IS PRINTED AND ENCLOSED HEREWITH. THE DISK HAS BOTH THE SOURCE CODE AND A 'BRUNABLE' COPY OF THE OVERVIEW.

AGAIN, THANKS FOR A TRULY USEFUL PUBLICATION.

MY BEST,



BROOKE (W BOERING)

PS. BILL MORGAN'S CATALOG ARRANGER WAS PARTICULARLY REVEALING OF THIS PART OF DOS. PLEASE THANK HIM FOR ME IF CONVENIENT.

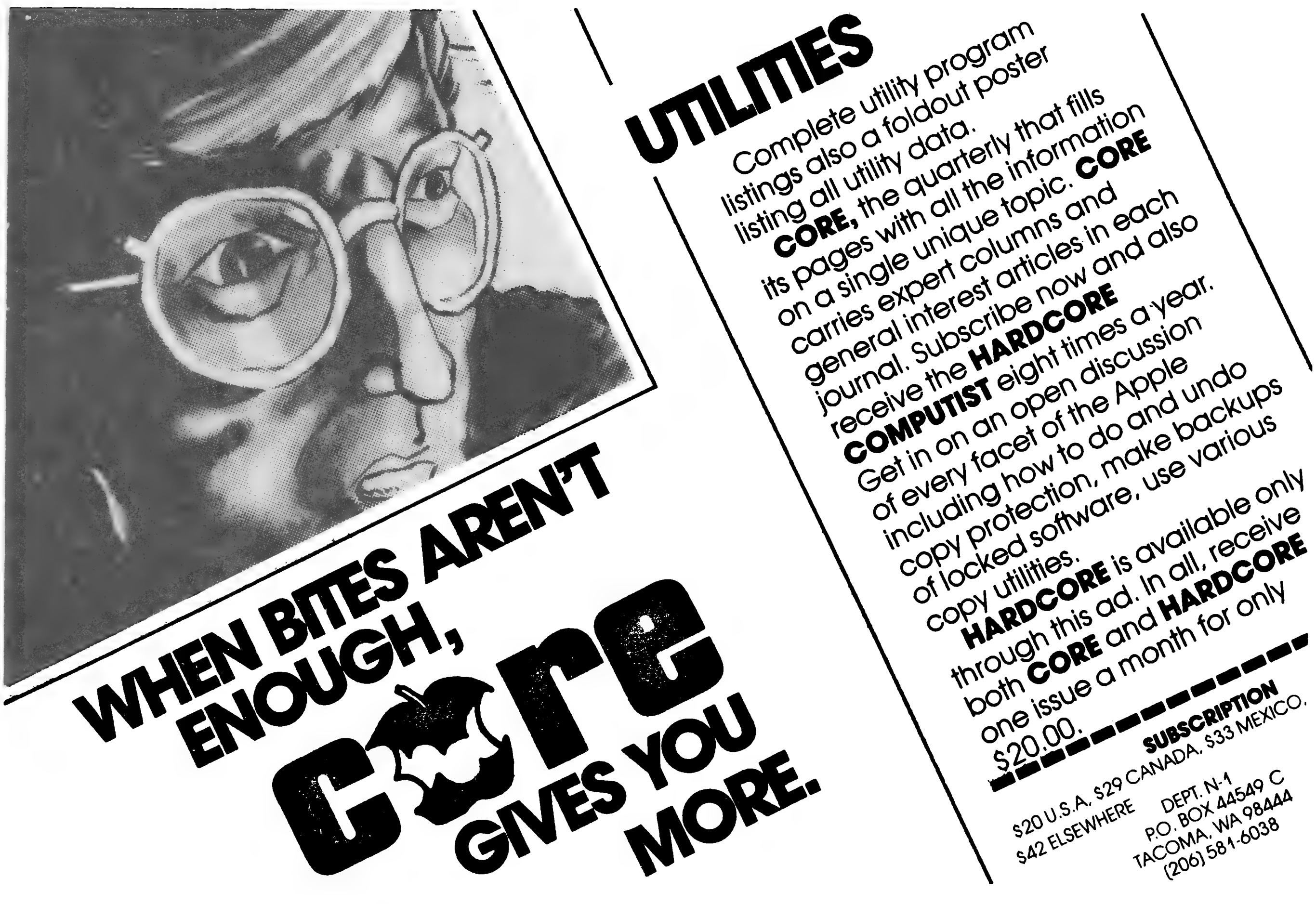
B

```
*--------------------------------------------------------------*
*    DATE PROCESSING MODULES                                   *
*      BY BROKE BOERING                                        *
*--------------------------------------------------------------*
          .OR $800
*--------------------------------------------------------------*
*    JUMP TABLE                                                *
*--------------------------------------------------------------*
*     JMP CONV1   MM/DD/YY  -> STD FMT
*     JMP CONV2   STD FMT   -> MM/DD/YY
*     JMP CONV3   STD FMT   -> CENTURY
*     JMP CONV4   DAY & WEEKDAY CODE
*                 RACK & STD FMT DATE UP
*                 (FROM 1 TO 225 DAYS)
*--------------------------------------------------------------*
*    MONITOR EQUATES                                           *
*--------------------------------------------------------------*
COUT      .EQ $FDED
PRBYTE    .EQ $FDDA
*--------------------------------------------------------------*
*    LOCAL EQUATES                                             *
*--------------------------------------------------------------*
LOC0      .EQ $40      (A3L)
LOC1      .EQ $41      (A3H)
LOC2      .EQ $42      (A5L)
LOC3      .EQ $43      (A5H)
ACL       .EQ $50
ACH       .EQ $51
XTNDL     .EQ $52
XTNDH     .EQ $53
AUXL      .EQ $54
AUXH      .EQ $55
ANSLO     .EQ $50
ANSHI     .EQ $51
PLIER     .EQ $52
CAND      .EQ $53
SAVER     .EQ $54
SLASH     .EQ $AF      (/)
*--------------------------------------------------------------*
*    LOCAL WORKING - - - - - - - - - - - - - - - - - - - - - - *
*--------------------------------------------------------------*
WKG       .HS 000000000000000000
BINYY     .EQ WKG+0
BINMM     .EQ WKG+1
BINDD     .EQ WKG+2
CENTURY.DAY.HI    .EQ WKG+4
CENTURY.DAY.LO    .EQ WKG+5
*--------------------------------------------------------------*
*    USER ALTERABLE CONTROLS                                   *
*--------------------------------------------------------------*
* LOWEST ACCEPTABLE YEAR
*   DEFAULT=75
* HIGHEST ACCEPTABLE YEAR
*   DEFAULT=88
* DAY-OF-WEEK SKIP
*   DEFAULT= SUNDAY & SATURDAY
*--------------------------------------------------------------*
*    CONVERT EXTERNAL FORMAT                                   *
*      MM/DD/YY TO STANDARD INTERNAL                           *
*      FORMAT: BITS YYYYYYYMMMMDDDDD                           *
*--------------------------------------------------------------*
* ENTRY: RA= DATA ADDRESS-LO
*        RY=      "     -HI
*  EXIT: CC= EQUAL IF OK
*            RA= YYYYYYYM BYTE
*            RX= MMMMDDDD BYTE
*            CC= NEQ IF ERROR
```

```
0800- 4C 14 08      JMP CONV1
0803- 4C BD 08      JMP CONV2
0806- 4C 07 09      JMP CONV3
0809- 4C 84 09      JMP CONV4

FDED-               COUT
FDDA-               PRBYTE

0040-               LOC0
0041-               LOC1
0042-               LOC2
0043-               LOC3
0050-               ACL
0051-               ACH
0052-               XTNDL
0053-               XTNDH
0054-               AUXL
0055-               AUXH
0050-               ANSLO
0051-               ANSHI
0052-               PLIER
0053-               CAND
00AF-               SLASH

080C- 00            WKG
080D- 00            BINYY
080E- 00            BINMM
080F- 00            BINDD
0810- 00            CENTURY.DAY.HI
0811- 00            CENTURY.DAY.LO
```

```
                        1640 CONV1
0814- 85 42             1650  STA LOC2        SET INDIRECT ADDR
0816- 84 43             1660  STY LOC3          :
0818- A9 00             1670  LDA #0          INIT WKG
081A- 8D 0D 08          1680  STA BINMM
081D- 8D 0E 08          1690  STA BINDD
0820- 8D 0C 08          1700  STA BINYY
                        1710 *-- DO 'MM'
0823- 20 7F 08          1720  JSR GET.DOUBLE
0826- D0 54             1730  BNE BADATE
0828- A8                1740  TAY             ZERO?
0829- F0 51             1750  BEQ BADATE
082B- C9 0D             1760  CMP #13         TOO HI?
082D- B0 4D             1770  BCS BADATE
082F- 8D 0D 08          1780  STA BINMM       ITS OK
0832- E6 42             1790  INC LOC2        KICK PAST '/'
                        1800 *-- DO 'DD'
0834- 20 7F 08          1810  JSR GET.DOUBLE
0837- D0 43             1820  BNE BADATE
0839- A8                1830  TAY             ZERO?
083A- F0 40             1840  BEQ BADATE
083C- AE 0D 08          1850  LDX BINMM       RX= INDEX TO LIST
083F- CA                1860  DEX
0840- DD F9 09          1870  CMP DAYS.COUNT,X
0843- 90 09             1880  BCC .3          G-A IF OK
0845- F0 07             1890  BEQ .3          G-A IF OK
0847- C9 1D             1900  CMP #29         29TH (OF FEB)?
0849- D0 31             1910  BNE BADATE      NO, ERR!
084B- 8C 0C 08          1920  STY BINYY       YES, SET YY-FLAG
                        1930 * (ACCEPT TEMPORARILY)
                        1940 .3
084E- 8C 0E 08          1950  STY BINDD       ITS OK (PROBABLY)
0851- E6 42             1960  INC LOC2        KICK PAST '/'
                        1970 *-- DO 'YY'
0853- 20 7F 08          1980  JSR GET.DOUBLE
0856- D0 24             1990  BNE BADATE
0858- CD F7 09          2000  CMP OLDEST.YEAR
085B- 90 1F             2010  BCC BADATE
085D- AE 0C 08          2020   LDX BINYY      RX= FEB 29TH FLAG
0860- 8D 0C 08          2030  STA BINYY       = 0YYYYYYY
0863- F0 04             2040   BEQ .6         G-A IF NOT FEB 29
0865- 29 03             2050  AND #$03        LEAP YEAR?
0867- D0 13             2060  BNE BADATE      ERR IF NOT LEAPYEAR
                        2070 *-- SET EXIT CONDITIONS
                        2080 .6
0869- AD 0D 08          2090  LDA BINMM
086C- 0A                2100  ASL
086D- 0A                2110  ASL
086E- 0A                2120  ASL
086F- 0A                2130  ASL
0870- 0A                2140  ASL
0871- 0D 0E 08          2150  ORA BINDD
0874- AA                2160  TAX             RX= MMMDDDDD
0875- AD 0C 08          2170  LDA BINYY
0878- 2A                2180  ROL             RA= YYYYYYYM
0879- A0 00             2190  LDY #0          EXIT OK
087B- 60                2200  RTS
                        2210
                        2220 BADATE
087C- A0 FF             2230  LDY #$FF       DATE ERROR EXIT
087E- 60                2240  RTS
```

```
                        2250 ********************************
                        2260 * S/R TO GET NEXT DOUBLE DIGIT
                        2270 *   (MAINLY USED FOR DATE INPUT)
                        2280 * ENTRY: LOC2/3= DATA ADDRESS
                        2290 GET.DOUBLE
087F- A0 00             2300  LDY #0
0881- B1 42             2310  LDA (LOC2),Y
0883- AA                2320  TAX             RX= TENS DIGIT
0884- E6 42             2330  INC LOC2
0886- B1 42             2340  LDA (LOC2),Y RA= UNITS DIGIT
0888- E6 42             2350  INC LOC2
088A- 20 8E 08          2360  JSR ASC2BIN
                        2370 * (CC= ERROR STATUS; PASS BACK)
088D- 60                2380  RTS
                        2390 ********************************
                        2400 * S/R TO CONVERT 2 ASCII DIGITS
                        2410 *   TO SINGLE BINARY BYTE
                        2420 *
                        2430 * ENTRY: RA= UNITS ASCII DIGIT
                        2440 *         RX= TENS ASCII DIGIT
                        2450 *
                        2460 *  EXIT: CC= EQUAL IF OK
                        2470 *           RA= BINARY EQUIV
                        2480 *         CC= NEQ IF NON DIGIT
                        2490 ASC2BIN
088E- 85 41             2500  STA LOC1        (SAVE TEMP)
0890- 8A                2510  TXA             RA= TENS
0891- C9 00             2520  CMP #0
0893- 90 25             2530  BCC NOTNUM
0895- C9 0A             2540  CMP #10
0897- B0 21             2550  BCS NOTNUM
0899- 29 0F             2560  AND #$0F
089B- F0 09             2570  BEQ .4
089D- AA                2580  TAX
089E- A9 00             2590  LDA #0
08A0- 18                2600  CLC
                        2610 .3
08A1- 69 0A             2620  ADC #10
08A3- CA                2630  DEX
08A4- D0 FB             2640  BNE .3
                        2650 .4
08A6- 85 40             2660  STA LOC0
08A8- A5 41             2670  LDA LOC1        RA= UNITS
08AA- C9 00             2680  CMP #0
08AC- 90 0C             2690  BCC NOTNUM
08AE- C9 0A             2700  CMP #10
08B0- B0 08             2710  BCS NOTNUM
08B2- 29 0F             2720  AND #$0F
08B4- 18                2730  CLC
08B5- 65 40             2740  ADC LOC0
08B7- A2 00             2750  LDX #0          SET EXIT= OK
08B9- 60                2760  RTS
                        2770
                        2780 NOTNUM
08BA- A2 FF             2790  LDX #$FF
08BC- 60                2800  RTS
```


WHEN BITES AREN'T ENOUGH, CORE GIVES YOU MORE.
UTILITIES
Complete utility program listings also a foldout poster listing all utility data.
CORE, the quarterly that fills its pages with all the information on a single unique topic. CORE carries expert columns and general interest articles in each journal. Subscribe now and also receive the HARDCORE COMPUTIST eight times a year.
Get in on an open discussion of every facet of the Apple including how to do and undo copy protection, make backups of locked software, use various copy utilities.
HARDCORE is available only through this ad. In all, receive both CORE and HARDCORE one issue a month for only $20.00.
SUBSCRIPTION
$20 U.S.A. $29 CANADA, $33 MEXICO.
$42 ELSEWHERE
DEPT. N-1
P.O. BOX 44549 C
TACOMA, WA 98444
(206) 581-6038

```
*-------------------------------
*  CONVERT STANDARD INTERNAL
*  DATE FORMAT, YYYYYYYMMMMDDDDD
*  TO EXTERNAL FORMAT MM/DD/YY.
*
*  ENTRY:  RA= HI BYTE (YYYYYYYM)
*          RX= LO BYTE (MMMDDDDD)
*          CV/CH PRESUMED PRESET
*-------------------------------
CONV2   JSR  EXPLODE TO BINYY,BINMM,BINDD
        LDA  BINMM   PRINT MM
        JSR  DATE.MM
        LDA  #'/'
        JSR  COUT
        LDA  BINDD   PRINT DD,
        JSR  DATE.DD
        LDA  #'/'
        JSR  COUT
        LDA  BINYY   PRINT YY
        JSR  DATE.YY
        RTS
*-------------------------------
*  S/R TO CONVERT YY BYTE TO DECI-
*  MAL,  THEN TO ASCII & DISPLAY.
*-------------------------------
DATE.YY CMP  #100    OVFLO PROTECT
        BCC  :4
        LDA  #99
:4      JMP  DATE.DD    GOTO COMMON
*-------------------------------
*  S/R TO CONVERT MM BYTE TO DECI-
*  MAL,  THEN TO ASCII & DISPLAY.
*-------------------------------
DATE.MM CMP  #12     OVFLO PROTECT
        BCC  :4
        LDA  #12
:4      JMP  DATE.DD    GOTO COMMON
*-------------------------------
*  S/R TO CONVERT DD BYTE TO DECI-
*  MAL,  THEN TO ASCII & DISPLAY.
*-------------------------------
DATE.DD LDX  #0      RX= 10'S CTR
        LDZ  #0
:2      CMP  #$A     < 10 ?
        BCC  :3      YES, JUMP OUT
        SEC
        SBC  #$A     MINUS 10
        INX          KICK 10'S CTR
        BNE  :2      LOOP BACK
*
:3      STA  LOCO    SAVE TEMP
        TXA          GET 10'S CTR
        ASL          POSN HI
        ASL
        ASL
        ASL
        ORA  LOCO    'OR' TOGETHER
        JMP  PRBYTE  PRINT IT
*RTS*
```

```
08BD-   20 D1 08
08C0-   4D 0D E6
08C3-   20 E9 AF
08C6-   20 49 ED
08C9-   20 0E FD
08CC-   AD 88 C0
08CF-   20 BB AD
08D2-   A9 EF 20
08D5-   ED AF A9
08D8-   20 ED FD
08DB-   AD 8C C0
08DE-   60
--------------
08DF-   C9 64 90
08E2-   02 A9 63
08E5-   4C EF 08
--------------
08E8-   C9 0C 90
08EB-   02 A9 0C
08EE-   4C EF 08
--------------
08F1-   A2 00 C9
08F4-   0A 90 06
08F7-   38 E9 0A
08FA-   E8 D0 F6
08FD-   60
--------------
08FE-   85 40 8A
0901-   0A 0A 0A
0904-   0A 05 40
0907-   4C DA FD
```

```
                                3440 *--------------------------------
                                3450 *      CONVERT STANDARD FORMAT TO
                                3460 *      CENTURY DAY & WEEKDAY CODE
                                3470 *
                                3480 * ENTRY: RA= YYYYYYYM
                                3490 *         RX= MMMMDDDD
                                3500 *
                                3510 *  EXIT: RA= CENTURY DAY (HI)
                                3520 *         RX= CENTURY DAY (LO)
                                3530 *         RY= WEEKDAY CODE
                                3540 *           1= MONDAY
                                3550 *           2= TUESDAY
                                3560 *           3= WEDNESDAY
                                3570 *           4= THURSDAY
                                3580 *           5= FRIDAY
                                3590 *           6= SATURDAY
                                3600 *           7= SUNDAY
                                3610 *           0= UNKNOWABLE
                                3620 CONV3
                                3630 *-- EXPLODE TO BINYY,BINMM,BINDD
0907- 20 D1 09                  3640  JSR EXPLODE.STANDARD.FORMAT
                                3650 *-- CALCULATE DAYS OF PRIOR YEARS
090A- AC 0C 08                  3660  LDY BINYY          STORE 256 DAYS
090D- 88                        3670  DEY                : FOR EACH
090E- 8C 10 08                  3680  STY CENTURY.DAY.HI : PRIOR YEAR
0911- 98                        3690  TYA                STORE 1 DAY
0912- 4A                        3700  LSR                : FOR EACH
0913- 4A                        3710  LSR                : PRIOR
0914- 8D 11 08                  3720  STA CENTURY.DAY.LO : LEAP YEAR
0917- A9 6D                     3730  LDA #109           STORE 109 DAYS
0919- 20 05 0A                  3740  JSR MULTIPLY.8X8   : FOR EACH
091C- 18                        3750  CLC                A  : PRIOR
091D- 6D 11 08                  3760  ADC CENTURY.DAY.LO : YEAR
0920- 8D 11 08                  3770  STA CENTURY.DAY.LO :
0923- 98                        3780  TYA                :
0924- 6D 10 08                  3790  ADC CENTURY.DAY.HI :
0927- 8D 10 08                  3800  STA CENTURY.DAY.HI :
                                3810
                                3820 *-- CALCULATE DAYS OF THIS YEAR
092A- AC 0E 08                  3830   LDY BINDD     RY= DD
092D- 98                        3840  TYA            (IN CASE WAS JAN)
092E- AE 0D 08                  3850  LDX BINMM      RX= MM
0931- CA                        3860  DEX            RX= MM-1
0932- F0 19                     3870  BEQ .7         G-A IF WAS JAN
0934- E0 01                     3880  CPX #1
0936- F0 08                     3890  BEQ .3         G-A IF WAS FEB
0938- AD 0C 08                  3900  LDA BINYY      (WAS MAR - DEC)
093B- 29 03                     3910  AND #$03       LEAP YEAR?
093D- D0 01                     3920  BNE .3         NO, G-A
093F- C8                        3930  INY            YES, KICK DAY CTR
                                3940 .3
0940- 98                        3950  TYA            RA= DD (OR DD+1)
                                3960 .4
0941- 18                        3970  CLC            ADD A MONTH'S DAYS
0942- 7D F8 09                  3980  ADC DAYS.COUNT-1,X :
0945- 90 03                     3990  BCC .5         G-A IF > 255 DAYS
0947- EE 10 08                  4000  INC CENTURY.DAY.HI
                                4010 .5
094A- CA                        4020  DEX            DECR CTR
094B- D0 F4                     4030  BNE .4         LOOP TIL DONE
                                4040 .7
                                4050 *-- ADD THIS YEAR'S DAYS
                                4060 *     TO PRIOR YEARS' DAYS
                                4070 *RA= DAYS THIS YEAR
094D- 18                        4080  CLC
094E- 6D 11 08                  4090  ADC CENTURY.DAY.LO
0951- 8D 11 08                  4100  STA CENTURY.DAY.LO
0954- 90 03                     4110  BCC .8
0956- EE 10 08                  4120  INC CENTURY.DAY.HI
                                4130 .8
                                4140 *-- CALCULATE WEEKDAY CODE
0959- AA                        4150  TAX              RX= CENTURY.DAY.LO
095A- AD 10 08                  4160  LDA CENTURY.DAY.HI
095D- 20 61 09                  4170  JSR GET.WEEKDAY
                                4180 *                 RY= WEEKDAY CODE
0960- 60                        4190  RTS
                                4200 *********************************
                                4210 *     CALCULATE WEEKDAY CODE FROM
                                4220 *     CENTURY DATE
                                4230 *
                                4240 * ENTRY: RA= CENTURY DATE-HI
                                4250 *         RX= CENTURY DATE-LO
                                4260 *
                                4270 *  EXIT: RA/RX= AS ON ENTRY
                                4280 *         RY= WEEKDAY CODE
                                4290 *           1= MONDAY
                                4300 *           2= TUESDAY
                                4310 *           3= WEDNESDAY
                                4320 *           4= THURSDAY
                                4330 *           5= FRIDAY
                                4340 *           6= SATURDAY
                                4350 *           7= SUNDAY
                                4360 *           0= UNKNOWABLE
                                4370 GET.WEEKDAY
0961- 85 51                     4380  STA ACH
0963- 86 50                     4390  STX ACL
0965- 48                        4400  PHA              SAVE RA
0966- 8A                        4410  TXA              SAVE RX
0967- 48                        4420  PHA                :
0968- A9 00                     4430  LDA #0             :
096A- 85 53                     4440  STA XTNDH        SET DIV'D (HIHI)
096C- 85 52                     4450  STA XTNDL        SET DIV'D (LOLO)
096E- 85 55                     4460  STA AUXH         SET DIVISOR(LO)
0970- A9 07                     4470  LDA #7           SET DIVISOR(HI)
0972- 85 54                     4480  STA AUXL           :
0974- A0 08                     4490  LDY #8           SET FOR 8BIT DIVSR
0976- 20 1E 0A                  4500  JSR DIVIDE.32X16
0979- A5 52                     4510  LDA XTNDL
097B- 18                        4520  CLC              REMAINDER + WEEKDAY
097C- 69 00                     4530  ADC #0           : OF 12/31/1900
097E- A8                        4540  TAY              (PRESET)
097F- 38                        4550  SEC
0980- E9 07                     4560  SBC #7
0982- 90 01                     4570  BCC .4           G-A IF RY OK
0984- A8                        4580  TAY              (RESET)
                                4590 .4
0985- C8                        4600  INY              ADJ: ANS+1 = CODE
0986- 68                        4610  PLA              RESTORE RA/RX
0987- 8A                        4620  TXA                :
0988- 68                        4630  PLA                :
0989- 60                        4640  RTS
```

```
                        4650 *--------------------------------
                        4660 *      ADD FROM 1 TO 225 DAYS TO
                        4670 *      A GIVEN STD FORMAT DATE
                        4680 *
                        4690 * ENTRY: RA= YYYYYYYM
                        4700 *         RX= MMMMDDDD
                        4710 *         RY= # DAYS TO ADD
                        4720 *  EXIT: RA/RX UPDATED
                        4730 CONV4
                        4740 *-- SAVE RY TO STACK
098A- 85 40             4750  STA LOC0
098C- 98                4760  TYA
098D- 48                4770  PHA
098E- A5 40             4780  LDA LOC0
                        4790 *-- EXPLODE TO BINYY,BINMM,BINDD
0990- 20 D1 09          4800  JSR EXPLODE.STANDARD.FORMAT
                        4810 *-- INIT FOR LOOP
0993- 68                4820  PLA             = # DAYS TO KICK
0994- 18                4830  CLC
0995- 6D 0E 08          4840  ADC BINDD       RA= WKG CTR
0998- AE 0D 08          4850  LDX BINMM       RX= WKG MM
                        4860 .2
                        4870 * IN THIS LOOP:
                        4880 *    RY= UTILITY REGISTER
                        4890 *    RX= WKG MM TO BE INCREMENTED
                        4900 *    RA= WKG CTR TO BE DECREMENTED
                        4910 * LOC3= WKG DAY COUNT FOR THE
                        4920 *       CURRENT MM (IN RX)
099B- BC F8 09          4930   LDY DAYS.COUNT-1,X
099E- 84 43             4940   STY LOC3        = MM'S DAY COUNT
09A0- E0 02             4950  CPX #2           IS MM FEB?
09A2- D0 0D             4960  BNE .4           NO, G-A
                        4970 *-- DO FEB
09A4- 48                4980  PHA             SAVE WKG CTR
09A5- AD 0C 08          4990  LDA BINYY
09A8- 29 03             5000  AND #$03        LEAP YEAR?
09AA- D0 04             5010  BNE .3           NO, G-A
09AC- A9 1D             5020  LDA #29          RESET DAY COUNT
09AE- 85 43             5030  STA LOC3         :
                        5040 .3
09B0- 68                5050  PLA             RESTORE WKG CTR
                        5060 .4
09B1- C5 43             5070  CMP LOC3
09B3- 90 12             5080  BCC .7           G-A IF DONE
09B5- F0 10             5090  BEQ .7           : (ALSO DONE)
09B7- 38                5100  SEC             WKG CTR MINUS
09B8- E5 43             5110  SBC LOC3        : WKG DAY COUNT
09BA- E8                5120  INX             MM+1
09BB- E0 0D             5130  CPX #13          OVFLO?
09BD- 90 DC             5140  BCC .2           NO, LOOP BACK
09BF- A2 01             5150  LDX #1           YES, SET MM= JAN
09C1- EE 0C 08          5160  INC BINYY       : AND SET YY+1
09C4- 4C 9B 09          5170  JMP .2          : AND LOOP BACK
                        5180 .7
09C7- 8D 0E 08          5190  STA BINDD
09CA- 8E 0D 08          5200  STX BINMM
09CD- 20 E6 09          5210  JSR IMPLODE.STANDARD.FORMAT
09D0- 60                5220  RTS

                        5230 ********************************
                        5240 * S/R TO EXPLODE STD FORMAT TO
                        5250 *   BINYY, BINMM & BINDD
                        5260 * ENTRY: RA= YYYYYYYM
                        5270 *         RX= MMMMDDDD
                        5280 *  EXIT: BINYY,BINMM,BINDD SET
                        5290 EXPLODE.STANDARD.FORMAT
09D1- 4A                5300  LSR             RA= 0YYYYYYY CC=M
09D2- 8D 0C 08          5310  STA BINYY
09D5- 8A                5320  TXA             RA= MMMDDDDD
09D6- 48                5330  PHA           SAVE MMMDDDDD
09D7- 6A                5340  ROR             RA= MMMMDDDD
09D8- 4A                5350  LSR                 0MMMMDDD
09D9- 4A                5360  LSR                 00MMMMDD
09DA- 4A                5370  LSR                 000MMMMD
09DB- 4A                5380  LSR                 0000MMMM
09DC- 8D 0D 08          5390  STA BINMM
09DF- 68                5400  PLA           PULL MMMDDDDD
09E0- 29 1F             5410  AND #$1F        RA= 000DDDDD
09E2- 8D 0E 08          5420  STA BINDD
09E5- 60                5430  RTS
                        5440 ********************************
                        5450 * S/R TO IMPLODE BINYY, BINMM &
                        5460 *   BINDD TO STD FORMAT
                        5470 * ENTRY: BINYY,BINMM,BINDD PRESET
                        5480 *  EXIT: RA= YYYYYYYM
                        5490 *        RX= MMMMDDDD
                        5500 IMPLODE.STANDARD.FORMAT
09E6- AD 0D 08          5510  LDA BINMM       RA= 0000MMMM
09E9- 0A                5520  ASL                 000MMMM0
09EA- 0A                5530  ASL                 00MMMM00
09EB- 0A                5540  ASL                 0MMMM000
09EC- 0A                5550  ASL                 MMMM0000
09ED- 0A                5560  ASL                 MMM00000 (CC=M)
09EE- 0D 0E 08          5570  ORA BINDD           MMMDDDDD
09F1- AA                5580  TAX             RX= MMMDDDDD (CC=M)
09F2- AD 0C 08          5590  LDA BINYY       RA= 0YYYYYYY (CC=M)
09F5- 2A                5600  ROL             RA= YYYYYYYM
09F6- 60                5610  RTS
                        5620 ********************************
                        5630
                        5640 OLDEST.YEAR
09F7- 4B                5650  .DA #75
                        5660 HIGHEST.YEAR
09F8- 54                5670  .DA #84
                        5680 DAYS.COUNT
09F9- 1F                5690  .DA #31 (JAN)
09FA- 1C                5700  .DA #28 (FEB)
09FB- 1F                5710  .DA #31 (MAR)
09FC- 1E                5720  .DA #30 (APR)
09FD- 1F                5730  .DA #31 (MAY)
09FE- 1E                5740  .DA #30 (JUN)
09FF- 1F                5750  .DA #31 (JUL)
0A00- 1F                5760  .DA #31 (AUG)
0A01- 1E                5770  .DA #30 (SEP)
0A02- 1F                5780  .DA #31 (OCT)
0A03- 1E                5790  .DA #30 (NOV)
0A04- 1F                5800  .DA #31 (DEC)
```

```
5810 ********************************************************************
5820 *                                                                  *
5830 *       8 X 8 MULTIPLY                                             *
5840 *                                                                  *
5850 *  ENTRY: RY= MULTIPLICAND                                         *
5860 *         RA= MULTIPLIER                                           *
5870 *                                                                  *
5880 *  EXIT:  RY= ANSWER-HI                                            *
5890 *         RA= ANSWER-LO                                            *
5900 *                                                                  *
5910 *  TIMING:  212 US - MAX                                           *
5920 *           180 US - MIN                                           *
5930 *           192 US - AVER                                          *
5940 *  NOTE: KEEP CLOSE TO SGN8X8                                      *
5950 *  MULTIPLY.8X8                                                    *
5960 MULTIPLIER STA  CAND     SAVE (MULTIPLIER)CAND
5970          CLR  RA         SET 8-BIT CTR
5980          LDA  #8
5990 MUL1     LSR  PLIER      TEST NEXT BIT
6000          BCC  MUL2       IF OFF, GO ROUND
6010          CLC
6020          ADC  CAND       IF ON, ADD
6030 MUL2     ROR  ANSLO      SHIFT ANSWER 1 BIT
6040          DEY             DECR POSITION CTR
6050          BNE  MUL1       LOOP TIL DONE 8 BITS
6060          TAY             RY= ANSWER-HI
6070          LDA  ANSLO      RA= ANSWER-LO
6080          RTS
6090 ********************************************************************
6100 *                                                                  *
6110 *       32 X 16 DIVIDE                                             *
6120 *                                                                  *
6130 *  PRE-ENTRY:                                                      *
6140 *     DIVIDEND IN:                                                 *
6150 *        XTNDH,XTNDL,ACH,ACL                                       *
6160 *     DIVISOR --> AUXL,AUXH                                        *
6170 *                                                                  *
6180 *  EXIT:  QUOTIENT -> ACL,ACH                                      *
6190 *         REMAINDER -> XTNDL,XTNDH                                 *
6200 DIV32X16 LDY  #$10       INDEX FOR 16 BITS
6210          .2
6220          ASL  ACL
6230          ROL  ACH        XTND/AUX
6240          ROL  XTNDL      : -> ACCUM
6250          ROL  XTNDH
6260          SEC
6270          LDA  XTNDL
6280          SBC  AUXL       MOD TO XTND.
6290          TAX
6300          LDA  XTNDH
6310          SBC  AUXH
6320          BCC  .3
6330          STX  XTNDL
6340          STA  XTNDH
6350          INC  ACL
6360          .3
6370          DEY
6380          BNE  .2
6390          RTS
6400 ********************************************************************
6410
```

```
0A05- 85 51
0A07- 84 52
0A09- A9 08
0A0B- A0

0A0D- 46 51
0A0F- 90 03
0A11- 18
0A12- 65 52
0A14- 64 50
0A16- 66
0A17- 88 F3
0A18- A8
0A1A- A5 50
0A1C- 60

0A1E- A0 10
0A20- 06 50
0A22- 26 51
0A24- 26 52
0A26- 38
0A27- A5 52
0A29- E5 54
0A2B- AA
0A2C- A5 53
0A2E- E5 55
0A30- 90 02
0A32- 86 53
0A34- 85 52
0A36- E6 50
0A38-
0A3A- 88 E3
0A3B- D0
0A3D- 60
```

N E W from Laumer Research
The S-C Macro Assembler Screen Editor.

Powerful Screen Editor for assembler files, co-resident with the S-C Macro Assembler allowing screen editing when you want it and S-C Macro Assembler editing too. Loads in the unused 4K bank of memory in a 16K Language Card.

Includes SYSGEN program for configuring standard 40 column Apple, 80 column VIDEX, or 80 column STB80 video drivers. Ajustable tabs, margins, horizontal and vertical scrolling, lines to 248 columns, and much more...

SOURCE code included. (Lets you learn about screen editors and configure for other brands of 80 column boards)

Based on a popular TI 990 editor for software developers. NOTE: this is not a word processor editor. Organized just for computer languages. If you work with assembly programs of 100 lines or more, then a Screen Editor is a MUST!

Requires 64K APPLE II with Language card and S-C Macro Assembler Language Card Version 1.0.

Price $49.00 from LAUMER RESEARCH
1832 SCHOOL RD.
CARROLLTON, TX 75006

Master Card and Visa accepted (send Name, card number and exp. date). Foreign orders add $3.00 shipping (US funds only).

The Apple][Circuit Description: A Review.....Bill Morgan

"Have you ever wanted to know the detailed circuit operation of your Apple][computer? Perhaps you were designing a peripheral or making a modification. Maybe you were repairing an Apple. You may have just been curious about how it works."

That's the first paragraph of a new book called The Apple][Circuit Description, by Winston D. Gayler. If the answer to that question is "yes", you need to look at this book. Circuit Description contains about 160 pages of text describing the operation of every component on the Apple's motherboard and keyboard. There are also 44 large fold-out pages of easy-to-read block diagrams, schematics, timing diagrams, and waveform drawings. The enlarged, readable schematics alone will be worth the price of the book to some users!

One of the first things Mr. Gayler handles is identifying the various revisions of the Apple][, from the original Rev. 0 through last year's RFI treated motherboard, Rev. D. The body of the book covers that last version, while an appendix goes into the differences in all earlier revisions, and the diagrams show all revisions. The very latest thing, the Apple //e, is not mentioned, since that's a radical departure from all others.

The book is intended for engineers, technicians, students, and serious hobbyists. The descriptions, schematics, timing diagrams, and waveform drawings can be an invaluable help in designing peripherals and modifications, troubleshooting, studying practical circuit design, and just understanding how your Apple works.

Each chapter has two sections, Overview, and Detailed Circuit Description. You can cruise the Overview sections to get an idea of what's going on in each piece of your Apple, or you can sit down with the Detailed Circuit Description, the schematics, your Apple, and your TTL Data Book, and figure out each and every signal in the computer.

Here is a chapter-by-chapter summary:

1. Introduction and overview of the book.

2. Block-diagram discussion of the whole computer's structure, introducing concepts like "address multiplexer" and "video address generator". Apple's unique patented power supply is also covered here.

3. Clocks: the master oscillator, clock generator, and the horizontal portion of the video address generator. Clocks are especially important in the Apple due to their interplay with the video circuitry.

4. The vertical portion of the video address generator and the sync, blanking, and color burst signals.

5. RAM memory, the 4116's and their addressing, as well as the shared access scheme for the video memory.

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

See these programs at participating Computerland and other fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

QUICKTRACE *$50*

Written by John Rogers

6. The 6502 processor and its internal cycles, including read cycles, write cycles, RAM and ROM cycles, I/O and keyboard cycles, interrupts, and DMA (direct memory access).

7. On-board I/O devices, including cassette I/O, the game port, the speaker, and the current two-piece keyboard.

8. Video generator hardware, how it creates TEXT, LORES, and HIRES displays under software control.

Appendices:

A. Introduction to standard video signal techniques, for those of us who know even less about video than about digital.

B. Various revisions of the Apple motherboard. The main text of the book describes the RFI, Rev. D board. This appendix covers the differences in all earlier boards, as well as the old one-piece keyboard.

C. Schematics. Pages and pages of enlarged diagrams of all versions of the motherboard and keyboards.

In the Introduction, Gayler says that the reader should be familiar with TTL (gates, flip-flops, shift registers, and multiplexers) and should have a basic knowledge of microprocessor and microcomputer architecture. Well, I have a very basic knowledge of architectures, and almost no familiarity with TTL details. This book looks like it will be a great tool for learning about TTL, because I will be able to relate what the data books say about a chip to a knowledge of what that chip is doing in my very own Apple.

One thing I would like to see is a sort of cross-reference by motherboard coordinate. It would be nice to be able to ask the book "What is the function of that 74LS20 at location D2?" As it is, I had to look through several foldouts for a chip symbol labelled "D2". It is a NAND gate in "Fig. C-2. Clock Generator (all revisions)" Since it's part of a clock circuit, it must be covered in chapter 3. Several minutes of poking around in chapter 3 tells me that chip is part of one of the Apple's most unique features! Every 65th CPU cycle is slightly stretched (1117 us vs. 978 us) to maintain sync with the color signals, and D2 is responsible for triggering that stretch.

That last paragraph started out to describe a shortcoming of the book, and turned into yet another example of the kind of great information contained in The Apple][Circuit Description. If you're doing any hardware work with the Apple, or if you want to learn more about what's going on in there, you need this book.

The Apple][Circuit Description, by Winston D. Gayler. Published by Howard W. Sams. 8 1/2 by 11 comb binding. 172 pp. text, 44 fold-out diagrams. Shipping weight 3 lbs. List price is $22.95, our price will be $21 + shipping ($2 domestic, $12 overseas).

New Version of DOS -- Patchers Beware

When Apple released the //e they apparently also slipped in a slightly revised version of DOS, called DOS 3.3e (or 3.3c. Reports differ.) The following information about the changes is from Tom Weishaar's DOStalk column in the April issue of Softalk.

The boot routine now throws a couple of new soft switches ($C00C and $C00E) and stores $FF in location $4FB. These steps turn off the //e's 80-column mode during boot-up.

A routine at $B331 that calculates position in a random access file is now simplified.

Now for the biggie: Another APPEND fix! (attempted) According to Weishaar, they eliminated a bug that occurred maybe once in 10,000 tries by introducing a new bug that bites once every 256 calls. Tom says that the most reliable method is to use the old DOS 3.3 and POKE -18851,0 before each APPEND.

The most significant thing about the APPEND change is where they put the patch: at $BA69! That used to be empty space and a popular place to install patches. No more! As a matter of fact, Bob's Fast Load patch in this issue goes into that area, and therefore should not be used with DOS 3.3e.

This means that //e users should be especially careful about installing published patches into DOS 3.3e, and all of us should quit using $BA69-BA95 for patches that will be distributed.

PATCHER: A General-Purpose Patch Installer.....Bill Morgan

My favorite new feature in Version 1.1 of the S-C Macro Assembler is the .PH directive. When Bob first described the new directive to me, I didn't quite see how to use it. Then he showed me a program like this one, and now I don't see how I did without it!

The directive .PH <expr> in an assembly causes the origin to be reset to <expr>, but the code continues to be stored in successive bytes of the same area as before. The result is much like the following lines all rolled into one:

```
2000 LABEL
2010         .OR SOMEWHERE.ELSE
2020         .TA LABEL
```

The difference is that the above lines would close an open Target File, whereas .PH SOMEWHERE.ELSE continues to direct code into the same file. The end of an offset block is marked with a .EP directive, that restores the origin to match the target address.

With this feature is so easy to assemble one program to create some patches and move them into place, all in one step. Anyway, here's the general purpose PATCHER, with some dummy code to show it off.

```
                        1000 *   S.PATCHER
                        1010 *--------------------------------
0000-                   1020 PNTR    .EQ $00,01
0002-                   1030 PATCH   .EQ $02,03
                        1040 *--------------------------------
                        1050         .OR $300
                        1060         .TF B.PATCHER
                        1070 *--------------------------------
                        1080 PATCHER
0300- A9 33             1090         LDA #PATCHES-1
0302- 85 00             1100         STA PNTR
0304- A9 03             1110         LDA /PATCHES-1
0306- 85 01             1120         STA PNTR+1
0308- A0 00             1130         LDY #0
                        1140
030A- 20 2B 03          1150 .1      JSR GET.BYTE LENGTH OF NEXT PATCH
030D- F0 1B             1160         BEQ .4       FINISHED
030F- AA                1170         TAX          SAVE LENGTH IN X
0310- 20 2B 03          1180         JSR GET.BYTE ADDRESS OF PATCH
0313- 85 02             1190         STA PATCH
0315- 20 2B 03          1200         JSR GET.BYTE
0318- 85 03             1210         STA PATCH+1
                        1220
031A- 20 2B 03          1230 .2      JSR GET.BYTE
031D- 91 02             1240         STA (PATCH),Y
031F- E6 02             1250         INC PATCH
0321- D0 02             1260         BNE .3
0323- E6 03             1270         INC PATCH+1
0325- CA                1280 .3      DEX
0326- D0 F2             1290         BNE .2
0328- F0 E0             1300         BEQ .1     ...ALWAYS
                        1310
032A- 60                1320 .4      RTS
                        1330 *--------------------------------
                        1340 GET.BYTE
032B- E6 00             1350         INC PNTR
032D- D0 02             1360         BNE .1
032F- E6 01             1370         INC PNTR+1
0331- B1 00             1380 .1      LDA (PNTR),Y
0333- 60                1390         RTS
                        1400 *--------------------------------
```

```
1000-           1410 P1.ORIGIN .EQ $1000
2000-           1420 P2.ORIGIN .EQ $2000
3000-           1430 P3.ORIGIN .EQ $3000
                1440
                1450 *  OTHER .EQUATES HERE
                1460
                1470 *--------------------------------
                1480 PATCHES
                1490
0334- 03 00 10  1500         .DA #P1.LENGTH,P1.ORIGIN
                1510         .PH P1.ORIGIN
                1520
                1530 PATCH1
                1540 *     PATCH1 CODE HERE
1000- 4C 00 20  1550         JMP PATCH2
                1560
0003-           1570 P1.LENGTH .EQ *-PATCH1
                1580         .EP
                1590 *--------------------------------
033A- 03 00 20  1600         .DA #P2.LENGTH,P2.ORIGIN
                1610         .PH P2.ORIGIN
                1620
                1630 PATCH2
                1640 *     PATCH2 CODE HERE
2000- 4C 00 30  1650         JMP PATCH3
                1660
0003-           1670 P2.LENGTH .EQ *-PATCH2
                1680         .EP
                1690 *--------------------------------
0340- 03 00 30  1700         .DA #P3.LENGTH,P3.ORIGIN
                1710         .PH P3.ORIGIN
                1720
                1730 PATCH3
                1740 *     PATCH3 CODE HERE
3000- 4C 00 10  1750         JMP PATCH1
                1760
0003-           1770 P3.LENGTH .EQ *-PATCH3
                1780         .EP
                1790 *--------------------------------
0346- 00        1800         .DA #0       END OF PATCHES
                1810 *--------------------------------

SYMBOL TABLE

032B- GET.BYTE    0002- PATCH
.01=0331          1000- PATCH1
0003- P1.LENGTH   2000- PATCH2
1000- P1.ORIGIN   3000- PATCH3
0003- P2.LENGTH   0300- PATCHER
2000- P2.ORIGIN   .01=030A, .02=031A, .03=0325, .04=032A
0003- P3.LENGTH   0334- PATCHES
3000- P3.ORIGIN   0000- PNTR
```

Notice that the object code columns show the bytes to be all over pages 3, 10, 20, and 30. The labels in the Symbol Table show the same thing. But, if you look around in memory, all this is in page 3. Once you type $300G, the JMP instructions will be moved to their true destinations.

Bob's DOS Fast Load patches elsewhere in this issue are an ideal example of how to use PATCHER. Here's all it takes:

1> Make the following changes to lines 1410-1430 of PATCHER

```
1410 P1.ORIGIN .EQ $BA69
1420 P2.ORIGIN .EQ $BCDF
1430 P3.ORIGIN .EQ $ACAF
```

2> Substitute Lines 1090-1160 of Fast Load for Line 1450 of PATCHER.

3> Substitute Lines 1210-1410 of Fast Load for Lines 1530-1550 of PATCHER.

4> Substitute Lines 1460-1590 of Fast Load for Lines 1630-1650 of PATCHER.

5> And substitute the following line for Lines 1740-1750 of PATCHER:

```
.DA PATCH1
```

Now you have a BRUNnable program which will quickly install the Fast Load patches into DOS. And if you want to add other DOS patches to the same program, just tack them in between lines 1790 & 1800.

If you want to patch something running in a RAM card, like the Macro Assembler, you just need to add the following lines:

```
1082          LDA $C083
1084          LDA $C083

1315 .4       LDA $C080
1320          RTS
```

And that's how I expect to handle patches from now on. Hope you find it useful!

More about the PRAWM Board.....Bob Sander-Cederlof

Advanced Peripheral Enterprises has introduced their PRAWM board, and is advertising elsewhere in this issue of AAL.

The PRAWM board contains from 2K to 8K of EEPROM. Data or programs can be written into the memory on the card, just as though it were RAM. Yet the memory is non-volatile, as in ROM, PROM, or EPROM. If you turn of your Apple, remove the card, ship it around the world...when you plug it back in the bytes will still be there!

EEPROM stands for "Electrically Eraseable PROM"; circuitry on the card allows you to individually write any bytes you wish, without erasing the rest of the memory. You do not need a separate EPROM programmer and ultraviolet EPROM eraser. There are no batteries either. The card is priced about the same as an EPROM card, but you save a lot of money on accessories. You will also save a lot of time, since you don't have to erase for 30-60 minutes, program chips for 5-20 minutes, and plug and unplug countless times. (You can program the entire 8K on a fully loaded PRAWM board in less than 25 seconds!)

The PRAWM card contains from 1 to 4 EEPROM chips, providing from 2K to 8K bytes. Each chip maps into the address space from $C800-$CFFF, and is accessed by switching in one chip at a time. On-board firmware makes it easy to move blocks of data between any chip and RAM.

By installing a jumper strap, you can even have the program stored in the first 2K chip automatically start up when you turn on your Apple, before or instead of booting a floppy. Just think of the possibilities: set up special commands, execute security procedures, power fail recovery, "boot" a mini-DOS of your own creation from PRAWM, eliminate the need for disk drives in turn-key monitoring applications...! Other strap options allow you to write-protect the board and to disable the $CFFF de-select function.

If you do a lot of development work involving EPROMs now, I think this card would be a big help. See Advanced Peripheral Enterprises' ad for price and ordering information.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).